CSE396, Spring 2026 Problem Set 9 Due Tue. 4/28, 11:59pm

This homework is due on **Tuesday**, April 28. Part (b) of problem (3) depends on the Thursday 4/23 lecture, but the TopHat part harks back to lectures already given.

Reading

Tuesday's lecture next week will cover the difficult "Reductions Via Computation Histories" subsection of the text's section 5.1. Here my notes (https://cse.buffalo.edu/~regan/cse396/CSE396S26Week14Tue.pdf) introduce a new kind of machine that also ties in the *skimmed* content of section 5.2: a "Two-Head DFA" (**2HDFA**). A 2HDFA can be pictured as having both heads on a single tape, initially scanning the first character of the input $x \in \Sigma^*$, or one head on each of two tapes that both hold a copy of x. They are not allowed to change any characters or move either head left. Transitions have the form $(p, c_1, c_2 \ / \ D_1, D_2, q)$ with the proviso that one or both of D_1, D_2 must be R for a right move. Also allowed are transitions with $c_1 = \sqcup$ (the blank at the end of x—or you can say \$ with the input given as $x\$$); then we must have $D_1 = S$ for stay and $D_2 = R$. Thus a 2HDFA on an input x of length n runs for at most $2n$ steps and then halts when both of its heads have processed x; the state in which this occurs determines whether it accepts or rejects x. The main point of a 2HDFA is that it is the weakest simple kind of automaton that can check the validity of computation traces by single-tape Turing machines. A 2HDFA is a very special case of a **deterministic linear bounded automaton** (DLBA), which the text employs for this purpose.

The lecture uses this picture of computation checking to segue to **computational complexity theory**, which is the subject of Chapter 7 of the text. (Chapter 6 is skipped.) The Thu. Apr. 30 lecture will finally introduce the famous classes P and NP and prove examples of **NP-completeness**. So read Chapter 7, except that at its end when hitting the proof of the Cook-Levin theorem starting on page 304, we will zoom instead to section 9.3 of chapter 9 and its "alternative proof" given as Theorem 9.34 on page 387. The proof there is much shorter and cleaner and relies only on your familiarity with the NAND gate and the fact that it is a universal logic gate. (The coverage there does not depend on sections 9.1–9.2, which can be skipped along with all of Chapter 8—the only space-copmplexity facts we will care about are that the regular languages constitute the class of constant "$O(1)$" space and the DLBAs equate to deterministic $O(n)$ space.)

——-**Homework**—part online and all *individual work*—due **Tue. 4/28, 11:59pm**——-

(1) Using *TopHat*, the "Worksheet" titled **Spr 226 HW9 Online Part**. Ten questions, each worth 2 points, for 20 total. The first five questions are intended as advance practice for the language classification problem (1) on the final exam.

(2) The *Turing Kit* shows Turing machine tapes that are infinite in both directions, but other sources stipulate a sharp left edge at cell 0. Some of those sources say that an attempted left move in cell 0 becomes a stay move, but let us say instead that the TM then *hangs*. This is a reasonable metaphor for real-world "hangs"—where unlike an infinite loop, you know immediately that something has gone wrong. We'd like to make both our Turing machines and our device drivers hang-free, but... So we have this decision problem:

- INSTANCE: A Turing machine M coded for a single one-way-infinite tape.
- QUESTION: Does there exist an input x such that $M(x)$ hangs?

Prove by reduction from any of A_{TM}, K_{TM}, or NE_{TM} (your choice) that this problem is undecidable. Show also that its language *is* computably enumerable. (*Hints:* A technical detail which you should reference in your proof: there is a guaranteed no-hang universal Turing machine U that the machine M' you create in your reduction can call for the "Simulate M on..." body of your code. Your M' will have other code that could be accompanied by the Kingston Trio recording of "Tom Dooley" [you can Google that]. 18 + 6 = 24 pts.)

(3) Define the language $INF = \{\langle M \rangle : L(M) \text{ is infinite}\}$.

(a) Prove by a reduction from K_{TM} that INF is not co-c.e. (Can you re-use the "all-or-nothing switch" reduction? 12 pts.)

(b) Prove by a reduction from D_{TM} that INF is not c.e. either—so INF is neither c.e. nor co-c.e. (Use the "delay switch" idea. 18 pts., for 30 on the problem and 74 on the set.)

Footnote about TMs with 1-way-infinite and 2-way-infinite tapes and makeing the former hang-proof: There are three ways to avoid hanging and/or simulate a Turing machine of the *Turing Kit* kind by a Turing machine whose single tape has a sharp left end.

(a) Stipulate that there is initially a $\wedge$ character in cell 0, so that inputs $x \in \Sigma^*$ are given in the form $\wedge x$ (or $\wedge x\$$ if you employ a right-endmarker too). The only instructions that read $\wedge$ have the form $(p, \wedge/\wedge, R, q)$, so that the machine always bounces off the $\wedge$ without hanging. Even without this, given input x flush-left, you can change the tape to $\wedge x$ by employing states r_c to "remember c" for each character $c \in \Sigma$, a "rewind state" w, and a new start state s' to hook into a machine that initially has the $\wedge$. The instructions for the preliminary "Shift-over" routing are: $(s, c \,/\, \wedge, R, r_c)$ for all c, $(r_c, d \,/\, c, R, r_d)$ for all $c, d \in \Sigma$ ($d = c$ allowed), $(r_c, \sqcup \,/\, c, L, w)$ for all $c \in \Sigma$, $(w, c \,/\, c, L, w)$ for all $c \in \Sigma$, and finally $(w, \wedge \,/\, \wedge, R, s')$.

(b) To simulate a machine M with a 2-way-infinite tape by M' with a 1-way infinite tape, one can start by doing (a). Then M' will thenceforth read the $\wedge$ exactly when M would move left to a blank cell it had not visited before. Then M' invokes a separate "Shift-Over" routine that changes its entire tape contents $\wedge y$ at any point to $\wedge\sqcup y$, ending back where it started with the head scanning the $\sqcup$ for the fresh blank cell created. For this to "caterpillar" all of y over correctly, it is important that M' itself never writes an actual blank $\sqcup$ internally—instead, it can use % as an alias for the blank character. (Exactly this is the "shift whole tape over daemon" routine used everywhere to make extra register space in the "Universal RAM Simulator" notes from the Thu. 4/2 lecture, except that it is triggered on '[' marking the right end of a register rather than $\wedge$.)

(c) The method in (b) needs $\Theta(|y|)$ steps by M' just to do one newly-leftward step by M, and so incurs quadratic slowdown. There is a smarter way to do this simulation with linear time overhead: Implement cell $+m$ on *Turing Kit* as cell $2m$ and cell $-m$ as cell $2m - 1$. Every time the *Turing Kit* machine moves a head R move your head R twice and similarly translate L to L twice, unless the move crosses cell 0, where you keep the $\wedge$ charcacter. If the original M wants to cross from right-of-cell-0 to left-of-cell-0 or vice-versa, then you have M' shift from using the "odd track" to the "even track"—or vice-versa. There is only a roughly-factor-of-2 time slowdown, and again the resulting machine is hang-free.